AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		FINAL Aug 88 - 31 Jul 91

4. TITLE AND SUBTITLE
"THEORY OF NEURAL NETWORKS" (U)

5. FUNDING NUMBERS
61102F
2305/K5

6. AUTHOR(S)
Dr. Yaser S. Abu-Mostafa

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
California Institute of Technology
Electrical Eng/Computer Science
Pasadena, CA 91125

8. PERFORMING ORGANIZATION REPORT NUMBER
AFOSR-TR- 92 0697

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM
Bldg 410
Bolling AFB DC 20332-6448

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
AFOSR-88-0213

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release;
Distribution unlimited

12b. DISTRIBUTION CODE
UL

13. ABSTRACT (Maximum 200 words)

A new neural unsupervised learning technique has been proposed in this work. Technique is based on the hierarcical partition of the patterns. Each partition corresponds to one neuron, which is in general a higher-order neuron. The partition is performed by iterating the neuron weights in an attempt to maximize a defined criterion function. The method is implemented on several examples and is found to give good results. In the second implemented example the method obtained a good solution, whereas the traditional adaptive resonance method and self-organizing maps produced unsatisfactory results. The method is fast, as it takes typically from about 2 to 5 iterations to coverage. Although the proposed method is prone to get stuck in local minima, this did happen in the simulations in only very difficult problems and this problem could be solved by using gradient algorithms for searching for the global maximum, like the Tunneling Algorithm.

14. SUBJECT TERMS	15. NUMBER OF PAGES
	27
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	SAR

Final Technical Report

THEORY OF NEURAL NETWORKS

Yaser S. Abu-Mostafa and Amir F. Atiya

Grant AFOSR 88-0213

submitted to:

Dr. Steven Suddarth
Air Force Office of Scientific Research
Bolling Air Force Base
Washington, DC 20332

Principal Investigator

Yaser S. Abu-Mostafa
Departments of Electrical Engineering and Computer Science
California Institute of Technology
Pasadena, California 91125

TABLE OF CONTENTS

I

THE VAPNIK CHERNOVENKIS DIMENSION: INFORMATION VERSUS COMPLEXITY IN LEARNING

I.1 INTRODUCTION

We start by formalizing a simple setup for learning from examples. We have an *environment* such as the set of visual images, and we call the set X. In this environment we have a *concept* defined as a function $f : X \to \{0,1\}$, such as the presence or absence of a tree in the image. The goal of learning is to produce a *hypothesis*, also defined as a function $g : X \to \{0,1\}$, that approximates the concept f, such as a pattern recognition system that recognizes trees. To do this, we are given a number of examples $(x_1, f(x_1)), \ldots, (x_N, f(x_N))$ from the concept, such as images with trees and images without trees.

In generating the examples, we assume that there is an unknown probability distribution P on the environment X. We pick each example independently according to this probability distribution. The statements in the paper hold true for any probability distribution P, which sounds very strong indeed. The catch is that the same P that generated the example is the one that is used to test the system, which is a plausible assumption. Thus we learn the tree concept by being exposed to 'typical' images. While X can be finite or infinite (countable or uncountable), we shall use a simple language that assumes no measure-theoretic complications.

The hypothesis g that we produce approximates f in the sense that g would rarely be significantly different from f [7]. This definition allows for two tolerance parameters ϵ and δ. With probability $\geq 1 - \delta$, g will differ from f at most ϵ of the time. The δ parameter protects against the small, but nonzero, chance that the examples happen to be very atypical.

A learning algorithm is one that takes the examples and produces the hypothesis. The performance is measured by the number of examples needed to produce a good hypothesis as well as the running time of the algorithm.

I.2 GENERALIZATION

We start with a simple case that may look at first as having little to do with what we think of as generalization. Suppose we make a blind guess of a hypothesis g, without even looking at any examples of the concept f. Now we take some examples of f and test g to find out how well it approximates f. Under what conditions does the behavior of g on the examples reflect its behavior in general?

This turns out to be a very simple question. On any point in X, f and g either agree or disagree. Define the agreement set

$$A = \{x \in X \ : \ f(x) = g(x)\}.$$

The question now becomes: How does the frequency of the examples in A relate to the probability of A? Let π be the probability of A, i.e., the probability that $f(x) = g(x)$ on a point x picked from X according to the probability distribution P. We can consider each example as a Bernoulli trial (coin flip) with probability π of success ($f = g$) and probability $1 - \pi$ of failure ($f \neq g$).

With N examples, we have N independent, identically distributed, Bernoulli trials. Let n be the number of successes (n is a random variable), and let $\nu = \frac{n}{N}$ be the frequency of success. Bernoulli's theorem states that, by taking N sufficiently large, ν can be made arbitrarily close to π with very high probability. In other words, if you take enough examples, the frequency of success will be a good estimate of the probability of success.

Notice that this does not say anything about the probability of success itself, but rather about how the probability of success can be estimated from the frequency of success. If on the examples we get 90% right, we should get about 90% right overall. If we get only 10% right, we should continue to get about the same. We are only predicting that the results of the experiment with the examples will persist, provided there are enough examples.

How does this case relate to learning and generalization? After all, we do not make a blind guess when we learn, but rather construct a hypothesis from the examples. However, at a closer look, we find that we make a guess, not of a hypothesis but of a set of hypotheses. For example, when the backpropagation algorithm [6] is used in a feedforward network, we are implicitly guessing that there is a good hypothesis among those that are obtained by setting the weights of the given network in some fashion. The set of hypotheses G would then be the set of all functions g that are obtained by setting the weights of the network in any fashion.

Therefore, when learning deals with a limited domain of representation, such as a given network with free weights, we in effect make a guess G of hypotheses. The learning

algorithm then picks a hypothesis $g \in G$ that mostly agrees with f on the examples. The question of generalization now becomes: Does this choice, which is based on the behavior on the examples, hold in general?

We can approach this question in a similar way to the previous case. We define, for every $g \in G$, the agreement set

$$A_g = \{x \in X \mid f(x) = g(x)\}.$$

These sets are different for different g's. Let π_g be the probability of A_g, i.e., the probability that $f(x) = g(x)$ on a point x picked from X according to the probability distribution P, for the particular $g \in G$ in question. We can again define random variables n_g (the number of successes with respect to different g's) and the frequencies of success $\nu_g = \frac{n_g}{N}$. At this point the problem looks exactly the same as the previous one and one may expect the same answer.

There is one important difference. In the simple Bernoulli case, the issue was whether ν converged to π. In the new case, the issue is whether the ν_g's converge to the π_g's *in a uniform manner* as N becomes large. In the learning process, we decide on one g but not the other based on the values of ν_g. If we had the ν_g's converge to the π_g's, but not in a uniform manner, we could be fooled by one erratic g. For example, we may be picking the hypothesis g with the maximum ν_g. With nonuniform convergence, the g we pick can have a poor π_g. We want the probability that *there is some* $g \in G$ such that ν_g differs significantly from π_g be very small. This can be expressed formally as

$$\Pr\left[\sup_{g \in G} |\nu_g - \pi_g| > \epsilon\right] \leq \delta.$$

where 'sup' denotes the supremum.

I.3 THE V-C DIMENSION

A condition for uniform convergence, hence generalization, was found by Vapnik and Chervonenkis [8]. The key is the inequality

$$\Pr\left[\sup_{g \in G} |\nu_g - \pi_g| > \epsilon\right] \leq 4m(2N)e^{-\epsilon^2 N/8},$$

where m is a function that depends on G. We want the RHS of the inequality to be small for large N, in order to achieve uniform convergence. The factor $e^{-\epsilon^2 N/8}$ is very helpful, since it is exponentially decaying in N. Unless the factor $m(2N)$ grows too fast, we should

be OK. For example, if $m(2N)$ is polynomial in N, the RHS will go to zero as N goes to infinity.

What is the function m? It depends on the set of hypotheses G. Intuitively, $m(N)$ measures the flexibility of G in expressing an arbitrary concept on N examples. For instance, if G contains enough hypotheses to be able to express any concept on 100 examples, one should not really expect any generalization with only 100 examples, but rather a memorization of the concept on the examples. On the other hand, if gradually more and more concepts cannot be expressed by any hypothesis in G as N grows, then the agreement on the examples means something, and generalization is probable. Formally, $m(N)$ measures the maximum number of *different* binary functions on the examples $x_1, \ldots, x_N$ induced by the hypotheses $g_1, g_2, \ldots \in G$.

For example, if X is the real line and G is the set of rays of the form $x \leq a$, i.e., functions of the form

$$g(x) = \begin{cases} 0 & x \leq a \\ 1 & x > a \end{cases},$$

then $m(N) = N + 1$. The reason is that on N points one can define only $N + 1$ different functions of the above form by sliding the value of a from left of the leftmost point all the way to right of the rightmost point.

There are two simple facts about the function m. First, $m(N) \leq |G|$ (where $|.|$ denotes the cardinality), since G cannot induce more functions that it has. This fact is useful only when G is a finite set of hypotheses. The second fact is that $m(N) \leq 2^N$, since G cannot induce more binary functions on N points than there are binary functions on N points. Indeed, there are choices of G (trivially the set of all hypotheses on X) for which $m(N) = 2^N$. For those cases, the V-C inequality does not guarantee uniform convergence.

The main fact about $m(N)$ that helps the characterization of G as far as generalization is concerned is that $m(N)$ is either identically equal to 2^N for all N, or else is bounded above by $N^d + 1$ for a constant d. This striking fact can be proved in a simple manner [4,8]. The latter case implies a polynomial $m(N)$ and guarantees generalization. The value of d matters only in how fast convergence is achieved. This is of practical importance because this determines the number of examples needed to guarantee generalization within given tolerance parameters.

The value of d turns out to be the smallest N at which G starts failing to induce all possible 2^N binary functions on any N examples. Thus, the former case can be considered the case $d = \infty$. d is called the V-C dimension [2,3].

I.4 INTERPRETATION

Training a network with a set of examples can be thought of as a process for selecting a hypothesis g with a favorable performance on the examples (large ν_g) from the set G. Depending on the characteristics of G, one can predict how this performance will generalize. This aspect of the characteristics of G is captured by the parameter d, the V-C dimension. If the number of examples N is large enough with respect to d, generalization is expected. This means that maximizing ν_g will approximately maximize π_g, the real indicator of how well the hypothesis approximates the concept.

In general, the more flexible (expressive, large) G is, the larger its V-C dimension d. For example, the V-C dimension of feedforward networks grows with the network size [2]. For example, the total number of weights in a one-hidden-layer network is an approximate lower bound for the V-C dimension of the network. While a bigger network stands a better chance of being able to implement a given function, its demands on the number of examples needed for generalization is bigger. These are often conflicting criteria. The V-C dimension indicates only the likelihood of generalization. This means, for better or for worse, whether the behavior on the examples is going to persist. The ability of the network to approximate a given function in principle is a separate issue.

The running time of the learning algorithm is a key concern [5,7]. As the number of examples increases, the running time generally increases. However, this dependency is a minor one. Even with few examples, an algorithm may need an excessive amount of time to manipulate the examples into a hypothesis. The independence of this complexity issue from the above discussion regarding information is apparent. Without a sufficient number of examples, no algorithm slow or fast can produce a good hypothesis. Yet a sufficient number of examples is of little use if the computational task of digesting the examples into a hypothesis proves intractable.

REFERENCES

[1] Y. S. Abu-Mostafa and D. Psaltis, "Optical Neural Computers," *Scientific American,* vol. 256, no. 3, pp. 88-95, 1987.

[2] E. B. Baum and D. Haussler, "What size network gives valid generalization," *Neural Computation,* MIT Press, vol. 1, pp. 151-160, 1989.

[3] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth, "Classifying learnable

geometric concepts with the Vapnik-Chervonenkis dimension," *Proc. ACM Symp. on Theory of Computing*, vol. 18, pp. 273-282, 1986.

[4] T. M. Cover, "Geometrical and statistical properties of systems of linear inequalities with applications in pattern recognition," *IEEE Trans. Electronic Computers*, pp. 326-334, June 1965.

[5] J. S. Judd, "On the complexity of loading shallow neural networks," *Journal of Complexity*, Academic Press, vol. 4, pp. 177-192, 1988.

[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing, vol. 1.* Cambridge, MA: MIT Press, 1986.

[7] L. G. Valiant, "A theory of the learnable," *Communications of the ACM,* vol. 27, pp. 1134-1142, 1984.

[8] V. N. Vapnik and A. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities," *Theory of Probability and Its Applications,* vol. 16, pp. 264-280, 1971.

II

LEARNING FROM HINTS IN NEURAL NETWORKS

II.1 INTRODUCTION

We can think of learning from examples as one end of a spectrum whose other end is explicit programming. Between these two extremes, there is a spectrum of largely unexplored possibilities.

To explain what we mean, let us assume that we have a decision function $f : X \rightarrow \{0,1\}$ that we wish to implement. For instance, the primality problem where $X = \{1,2,3,\ldots\}$ and $f(x) = 1$ iff x is a prime. Also, the problem of recognizing a tree in an image where X is a set of images and $f(x) = 1$ iff x contains a tree. The goal is to come up with an implementation of f. We can write a simple program for f in the primality problem. However, the lack of a mathematical understanding of f in the image recognition problem forces us to seek other approaches. One approach is learning from examples, where we use a 'learning process' that we present with examples of images with trees and images without trees until the process infers an implementation of f. Whenever we have an effective process for learning from examples, it is tantamount to automated programming. The process is a mechanical means of producing an implementation of f.

When feasible, learning from examples is a very convenient approach. It does not require any knowledge of f, just input-output examples. In many practical situations, we do have some knowledge of f. In these cases, it would be inefficient to take blind examples without taking advantage of what we already know about f. This gives rise to learning from hints as opposed to learning from examples. Learning from hints is still a *learning* process, since we do not know enough about f to program it outright.

A hint is any piece of information about f. As a matter of fact, an input-output example is a special case of a hint. A hint may take the form of a global constraint on f, such as a symmetry property or an invariance. It may also be partial information about the implementation of f.

A hint may be valuable to the learning process in two ways. It may reduce the number of functions that are candidates to be f (information value), and may reduce the number of steps needed to find the implementation of f (complexity value). For illustration, suppose we are learning about an unknown integer N ($10^5 < N < 10^6$) and we want to represent

it as a six-digit number. Which is a more valuable hint: *The number is a prime* or *The most significant digit is 7* ? Although the first hint has more information value, it may have less complexity value because it does not reduce the search space in a way that is easily compatible with the desired representation.

We shall report a positive result and a negative result in this paper. The positive result is a technique that incorporates any invariance hint in any descent technique for learning. The negative result is that general learning in neural networks remains NP-complete even with a hint that is biologically plausible. This paper provides a preliminary treatment of the subject of hints, and many directions of investigation warrant further exploration.

For completeness, we briefly introduce feedforward neural networks and their gradient descent learning. A single-output feedforward neural network (Figure II.1) is a combinational circuit organized in layers of units (neurons). Each neuron performs a threshold function $\theta(\sum_j w_j u_j - t)$, where $\{u_j\}$ are the inputs to the neuron, $\{w_j\}$ are real numbers (weights), t is a real number (threshold), and θ is a 'sigmoid' function (soft or hard) that varies between -1 and $+1$ monotonically (the binary convention is ± 1 instead of $0, 1$).

For any set of values for the weights and the thresholds, the network output y is a fixed function of the input variables $\mathbf{x} = x_1 \ldots x_N$. In order to make the network implement a function $f(\mathbf{x})$, we need to choose the weights and the thresholds such that the actual output y and the desired output $f(\mathbf{x})$ are as close as possible. A gradient descent method for learning f from examples minimizes $(y - f(\mathbf{x}))^2$ for each example by perturbing the weights and thresholds. The formula for perturbing the weights is

$$\Delta w_i \propto -\frac{\partial}{\partial w_i}(y - f(\mathbf{x}))^2 = -2(y - f(\mathbf{x}))\frac{\partial y}{\partial w_i} .$$

When this formula is applied to the neurons of a feedforward network, the resulting rule is the backpropagation algorithm of Werbos, described in (Rumelhart, Hinton, and Williams, 1986).

II.2 INVARIANCE HINTS

Many of the hints we have in pattern recognition problems are invariance hints. A hand written letter S can be deformed in many ways without losing its identity. Properties such as shift invariance, scale invariance, and rotation invariance are commonplace in image recognition.

An invariance property can be formalized as a set $\mathcal{A}$ of subsets $A \subset X$ such that if $x_1, x_2 \in A$, then $f(x_1) = f(x_2)$. Thus f is invariant within each set $A \in \mathcal{A}$. Interesting

invariances are usually common to many functions f on the same domain X. For example, if X is the set of images, many recognition functions share some form of scale invariance. In this case, each set $A \in \mathcal{A}$ consists of images which are scaled versions of one another.

How can the invariance hint help the learning process? One way is to incorporate the hint directly in the implementation. For illustration, consider the implementation of a function $f : \{-1,+1\}^N \rightarrow \{-1,+1\}$ on a feedforward neural network. If we know that f is an even function, i.e., $f(x_1, x_2, \ldots, x_N) = f(-x_1, -x_2, \ldots, -x_N)$, we can incorporate this hint in the network as shown in Figure II.2.

Each input x_i is applied to two sets of neurons, with one set implementing the dual functions of the other set. The dual of $f(x_1, x_2, \ldots, x_N)$ is defined as $-f(-x_1, -x_2, \ldots, -x_N)$, and can be implemented by using the same weights and the negative of threshold used to implement f. The outputs of the two dual neurons are then combined into the neurons of the next layer using a weight and its negative. From then on, the function implemented by the network is forced to be even in the variables $x_1, \ldots, x_N$. These constraints on the neurons of the first two layers must be taken into consideration in the learning process when examples of f are given. For instance, if we apply gradient descent, we cannot treat all the weights as independent variables any more.

Being an even function is much simpler than the invariance hints we are likely to encounter in pattern recognition. It may not be as easy to come up with a structure of the network that automatically guarantees a complicated invariance, such as elastic deformation. We will develop a unified method for incorporating any invariance, not in the structure of the network, but rather in the gradient descent method itself.

The key idea is expressing the hint itself as a set of examples. Suppose we are dealing with a function $f : \{-1,+1\}^N \rightarrow \{-1,+1\}$, which is invariant under cyclic shift of the input bits, e.g., $f(\mathbf{x}_1) = f(\mathbf{x}_2)$, where

$$\mathbf{x}_1 = -1\ -1\ +1\ +1\ -1\ +1\ -1\ +1\ -1\ -1\ -1\ -1\,,$$
$$\mathbf{x}_2 = -1\ -1\ -1\ -1\ +1\ +1\ -1\ +1\ -1\ +1\ -1\ -1\,.$$

The condition $f(\mathbf{x}_1) = f(\mathbf{x}_2)$ is an example of the hint as much as $f(\mathbf{x}) = +1$ would be an example of the function. Just like $f(\mathbf{x}) = +1$ can be enforced as a minimization of $(y-1)^2$, where y is the output of the network when the input is $\mathbf{x}$, $f(\mathbf{x}_1) = f(\mathbf{x}_2)$ can be enforced as a minimization of $(y_1 - y_2)^2$, where y_1 and y_2 are the outputs of the network when the inputs are $\mathbf{x}_1$ and $\mathbf{x}_2$, respectively. The gradient descent perturbation of the weights in this case would be

$$\Delta w_i \ \propto\ -\frac{\partial}{\partial w_i}(y_1 - y_2)^2 \ =\ -2(y_1 - y_2)\left(\frac{\partial y_1}{\partial w_i} - \frac{\partial y_2}{\partial w_i}\right)\,,$$

which is similar to the perturbation due to two examples of f.

The same idea is valid for all invariances and all descent techniques. It makes it possible to incorporate in a regular algorithm for learning from examples what would otherwise be a hard-to-implement invariance. The initial layers of the network can be dedicated to learning the invariance (the learning counterpart of the approach of Figure II.2).

A look at the expression for Δw_i reveals that *we do not need the actual value of* f in order to compute the perturbation of the weights due to an example of the invariance hint. This makes it possible to generate an arbitrary number of (possibly artificial) examples of the hint for the learning process without having to compute f (or even without the need to know which f we are learning). Such resource may be valuable if we have a limited number of natural examples of f.

This observation can be formalized within the framework of Vapnik and Chervonenkis (1971). If the set of candidate functions is significantly reduced by the constraint that they must satisfy the invariance property, the number of examples of f needed for the learning process decreases accordingly (Abu-Mostafa, 1989).

II.3 COMPLEXITY ISSUES

The process of learning an unknown function f in a feedforward network can be considered a search in Euclidean space for a set of weights and thresholds that implements the function. Indeed, without restricting the domain of functions and networks, the learning problem is NP-complete (Judd, 1988). Other, apparently simplistic, learning problems have also been shown to be NP-complete (Valiant, 1984).

Does the incorporation of hints in the learning process reduce the time complexity of learning? It is plausible that a hierarchical decomposition of the search space that results from learning different hints independently will reduce the search time. However, we were unable to show a meaningful instance of a hint that rendered an NP-complete learning problem polynomial-time. We will report on an interesting hint that did not change the NP-completeness of the problem.

Consider the general problem of learning in a feedforward network with hard thresholds. Assume that, as part of the input to the learning process, we are given the *signs* of a set of weights that does implement the function in question. This hint is biologically motivated. In actual neurobiological systems, certain synapses are predisposed to be inhibitory and others to be excitory. Only the magnitude of the weight, not the sign, is left to the learning process.

The problem remains NP-complete as it turns out to be polynomially related to the old problem. To see this, we replace each synapse of the old network by the subnetwork of Figure II.3a. There are clearly choices of the weights with the prescribed signs that leads to an arbitrary equivalent weight for the original synapse.

It is interesting to note that the problem also remains NP-complete if we are given the absolute values of the weights and need only to find the signs! The reason is that each synapse can be replaced by the subnetwork of Figure II.3b. The prescribed moduli of the weights in this subnetwork can be given a pattern of signs that leads to an arbitrary equivalent weight (of finite accuracy) for the original synapse. Since we never need more than a polynomial number of bits for the weight (Hong, 1987), the polynomial equivalence is established.

II.4 CONCLUSION

Hints are pieces of information about an unkown function that we wish to learn, ranging from input-output examples to a complete implementation of the function. Invariance hints can be incorporated into descent methods of learning from examples, with possible gains in information and complexity. Certain strong hints do not change the NP-completeness status of learning.

Several directions of investigating the subject of hints remain open. To name a few, the compatibility of the hints with the desired implementation of f and with each other and how this affects their complexity value, the quantification of the information value of a hint in terms of the change in the V-C dimension, and finding natural examples of NP-complete learning problems that become polynomial-time using plausible hints.

ACKNOWLEDGEMENT

I would like to thank Dr. Robert Snapp for his assistance.

REFERENCES

Abu-Mostafa, Y. S. (1988), Random problems, *Journal of Complexity* **4**, 277-284.

Abu-Mostafa, Y. S. (1989), The Vapnik-Chervonenkis dimension: information versus complexity in learning, *Neural Computation* **1**.

Hong, J. (1987), "On Connectionist Models," Technical Report (Computer Science), *Uni-*

versity of Chicago, Chicago, IL.

Judd, J. S. (1988), On the complexity of loading shallow neural networks, *Journal of Complexity* **4**, 177-192.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986), Learning internal representations by error propagation, *in* "Parallel Distributed Processing, vol. 1," pp. 318-362, MIT Press, Cambridge, MA.

Valiant, L. G. (1984), A theory of the learnable, *Comm. of the ACM* **27**, 1134-1142.

Vapnik, V. N., and Chervonenkis, A. (1971), On the uniform convergence of relative frequencies of events to their probabilities, *Theory of Probability and Its Applications* **16**, 264-280.

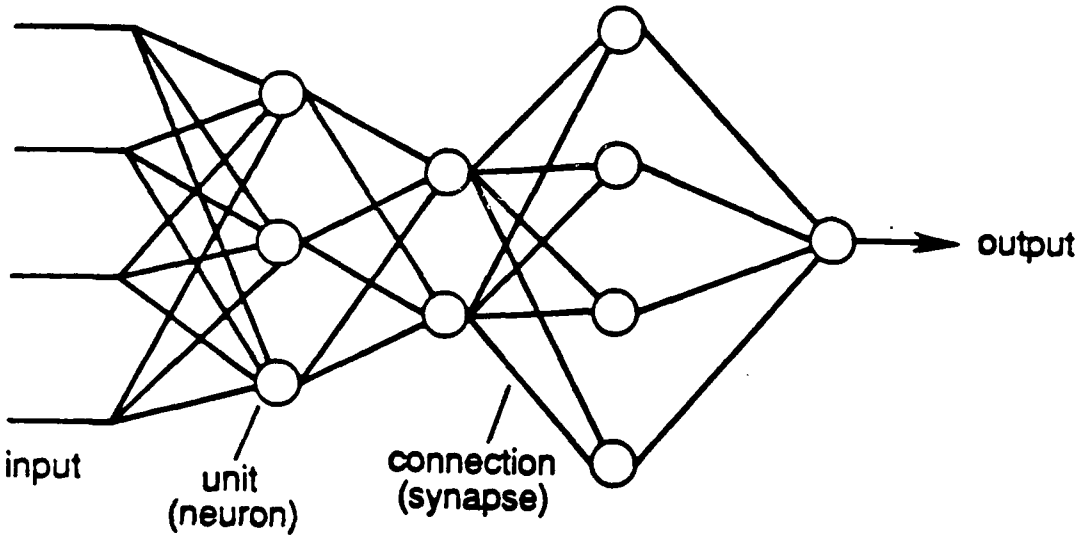

Figure II.1: Feedforward neural network

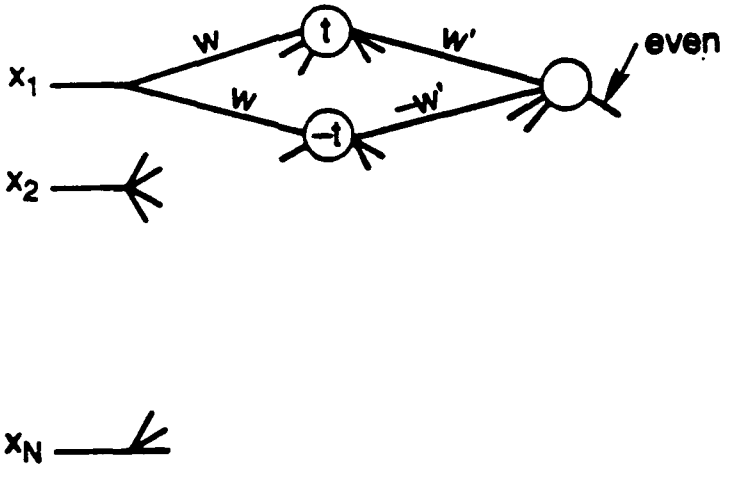

Figure II.2: Network for even functions

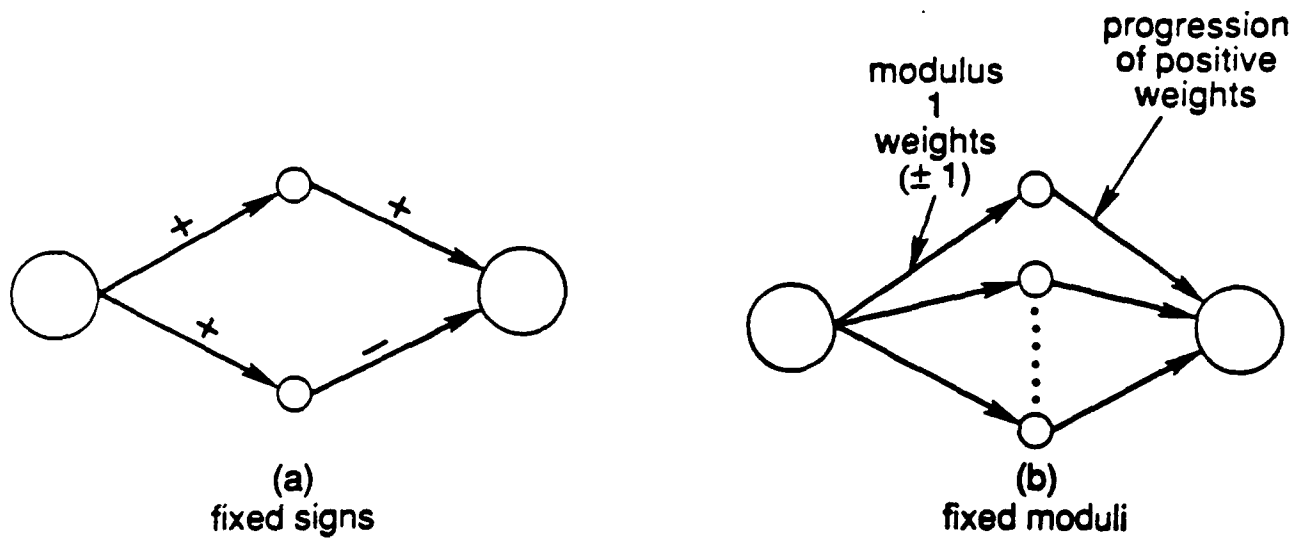

Figure II.3: Simulating arbitrary weights

III

AN UNSUPERVISED LEARNING TECHNIQUE FOR ARTIFICIAL NEURAL NETWORKS

III.1 INTRODUCTION

One of the most important features in neural networks is its learning ability, which makes it in general suitable for computational applications whose structure is relatively unknown. Pattern recognition is one example of such kinds of problems. For pattern recognition there are mainly two types of learning: supervised and unsupervised learning (refer to [1]). For supervised learning the training set consists of typical patterns whose class identities are known. For unsupervised learning, on the other hand, information about the class membership of the training patterns is not given, either because of lack of knowledge or because of the high cost of providing the class labels associated with each of the training patterns.

A number of neural network models for unsupervised learning have been proposed, for example Grossberg's adaptive resonance [2] and Kohonen's self-organizing maps [3]. In pattern recognition problems the pattern vectors tend to form clusters, a cluster for each class, and therefore the first step for a typical unsupervised learning technique is the estimation of these clusters. The clusters are usually separated by regions of low pattern density. We present here a new method for unsupervised learning in neural networks (see also [4]). The basic idea of the method is to update the weights of the neurons in a way to move the decision boundaries in places sparse in patterns. This is one of the most natural ways to partition clusters. It is more or less similar to the way humans decompose clusters of points in three or less dimensions visually. This way contrasts with the traditional approach of the adaptive resonance and the self-organizing maps, whereby the estimation of the membership of a pattern vector depends only on respectively the inner products and the Euclidean distances between this vector and suitably estimated class representative vectors.

III.2 THE MODEL

Let $\mathbf{x}^{(1)}, ..., \mathbf{x}^{(N)}$ represent the training pattern vectors. Let the dimension of the vectors be n. The training patterns represent several different classes. The class identities

of the training patterns are not known. Our purpose is to estimate the number of classes available as well as the parameters of the neural classifier. Let us first consider the case of having two classes. Our neural classifier consists of one neuron (see Fig. III.1) with n inputs corresponding to the n components of the pattern vector $\mathbf{x}$ to be classified. The output of the neuron is given by

$$y = f(\mathbf{w}^T\mathbf{x} + w_o)$$

where $\mathbf{w}$ is a weight vector, w_o is a threshold and f is a sigmoid function from -1 to 1 with $f(0) = 0$, e.g.

$$f(u) = \frac{1 - e^{-u}}{1 + e^{-u}}.$$

Positive output means class 1, negative output means class 2 and zero output means undecided. The output can be construed as indicating the degree of membership of the pattern to each of the two classes; thus we have a fuzzy classifier.

The decision boundary is the hyperplane

$$\mathbf{w}^T\mathbf{x} + w_o = 0$$

(see Fig. III.2). Patterns near the decision boundary will produce outputs close to zero while patterns far away from the boundary will give outputs close to 1 or -1. Assuming $\|\mathbf{w}\| \leq a$, where a is a constant, a good classifier is one which produces an output close to 1 or -1 for the training patterns, since this indicates its "decisiveness" with respect to the class memberships. Therefore, we design the classifier so as to maximize the criterion function:

$$J = \frac{1}{N}\sum_j f^2(\mathbf{w}^T\mathbf{x}^{(j)} + w_o) - [\frac{1}{N}\sum_j f(\mathbf{w}^T\mathbf{x}^{(j)} + w_o)]^{2q}$$

subject to $\|\mathbf{w}\| \leq a$, where q is a positive integer (for best results we take q=2). The first term is a measure of the decisiveness of the classifier with the given test pattern set. Regarding the second term, it has the following purpose. The first term is maximized if the decision hyperplane is very far away from the patterns resulting in the output being very close to 1 for all test patterns (or close to -1 for all patterns), i.e. all patterns are assigned to one class only, which is a trivial solution. Incorporating the second term will prevent this trivial solution because if $f(\mathbf{w}^T\mathbf{x}^{(j)} + w_o) \simeq 1$ for all j (or -1 for all j), then J will be nearly zero. However, J is non-negative for any $\mathbf{w}$ and w_o because of the following,

$$J = \frac{1}{N}\sum_j f^2(\mathbf{w}^T\mathbf{x}^{(j)} + w_o) - (\bar{f})^{2q}$$

$$\geq \frac{1}{N}\sum_j f^2(\mathbf{w}^T\mathbf{x}^{(j)} + w_o) - (\bar{f})^2$$

where

$$\bar{f} = \frac{1}{N}\sum_j f(\mathbf{w}^T\mathbf{x}^{(j)} + w_o).$$

The inequality follows because $\bar{f}$, the average of the outputs, is less than one in magnitude and $q \geq 1$. Then we get

$$J \geq \frac{1}{N}\sum_j \left[f(\mathbf{w}^T\mathbf{x}^{(j)} + w_o) - \bar{f}\right]^2 \geq 0$$

and hence the trivial solution of assigning all patterns to one class only is ruled out because it corresponds to a minimum not a maximum for J.

We iterate the weights in a steepest ascent manner in an attempt to maximize the defined measure,

$$\begin{aligned}\Delta\mathbf{w} &= \rho\frac{\partial J}{\partial \mathbf{w}} \\ &= \rho\frac{2}{N}\sum_j \left[f(\mathbf{w}^T\mathbf{x}^{(j)} + w_o) - q(\bar{f})^{2q-1}\right] f'(\mathbf{w}^T\mathbf{x}^{(j)} + w_o)\mathbf{x}^{(j)}\end{aligned}$$

and

$$\begin{aligned}\Delta w_o &= \rho\frac{\partial J}{\partial w_o} \\ &= \rho\frac{2}{N}\sum_j \left[f(\mathbf{w}^T\mathbf{x}^{(j)} + w_o) - q(\bar{f})^{2q-1}\right] f'(\mathbf{w}^T\mathbf{x}^{(j)} + w_o)\end{aligned}$$

where ρ is the step size. The iteration is subject to the condition $\|\mathbf{w}\| \leq a$. Whenever after some iteration this condition is violated, $\mathbf{w}$ is projected back to the surface of the hypersphere $\|\mathbf{w}\| = a$ (simply by multiplying by $a/\|\mathbf{w}\|$). The term $f'(\mathbf{w}^T\mathbf{x}^{(j)} + w_o)$ in the update expression gives the patterns near the decision boundary more effect than the patterns far away from the decision boundary. This is because $f'(u)$ is high whenever u is near zero and goes to zero when the magnitude of u is large. Thus, what the method is essentially doing is iterating the weights in a way to move the decision boundary to a place sparse in patterns. The relative importance of patterns near the decision boundary increases for large a. This is because $\mathbf{w}$ tends to go to the surface of the sphere $\|\mathbf{w}\| \leq a$. Large a will therefore result in the argument of f' being in general higher than for the case of small a, thus resulting in a smaller strip with high f' around the decision boundary. Of course without the constraint $\mathbf{w}$ could grow in an unbounded fashion, resulting in $f(\mathbf{w}^T\mathbf{x}^{(j)} + w_o)$ being very near 1 or -1 for all j for most possible partitions and we obtain therefore possibly bad solutions.

III.3 EXTENSIONS

We have shown in the previous section a method for training a linear classifier. To extend the analysis to the case of a curved decision boundary we use a higher order neuron, i.e. the output is described by

$$y = f(w_o + \sum_i w_i x_i + \cdots + \sum_{i_1 \leq \cdots \leq i_L} w_{i_1 \cdots i_L} x_{i_1} \cdots x_{i_L})$$

where x_i denotes the i^{th} component of the vector $\mathbf{x}$ and L represents the order. Higher order networks have been investigated by several researchers, refer for example to [5],[6] and [7]. They achieve simplicity of design while still having the capability of producing fairly sophisticated non-linear decision boundaries.

The decision boundary for the higher order neuron is given by the equation

$$w_o + \sum_i w_i x_i + \cdots + \sum_{i_1 \leq \cdots \leq i_L} w_{i_1 \cdots i_L} x_{i_1} \cdots x_{i_L} = 0.$$

The higher order case is essentially a linear classifier applied to augmented vectors of the form

$$(x_1, \cdots, x_n, \cdots, x_1^L, x_1^{L-1} x_2, \cdots, x_n^L)$$

(the superscripts denote here powers) using an augmented weight vector

$$(w_1, \cdots, w_n, \cdots, w_{1 \cdots 11}, w_{1 \cdots 12}, \cdots, w_{n \cdots nn}).$$

We can therefore apply basically the same training procedure described for the linear classifier case on the set of augmented vectors, i.e. we update the weights according to

$$\Delta w_{i_1 \cdots i_l} = \rho \frac{2}{N} \sum_j [f(u_j) - q(\bar{f})^{2q-1}] f'(u_j) x_{i_1}^{(j)} \cdots x_{i_l}^{(j)}$$

$$\Delta w_o = \rho \frac{2}{N} \sum_j [f(u_j) - q(\bar{f})^{2q-1}] f'(u_j)$$

where $f(u_j)$ is the output of the neuron when applying the pattern $\mathbf{x}^{(j)}$, i.e.

$$u_j = w_o + \sum_i w_i x_i^{(j)} + \cdots + \sum_{i_1 \leq \cdots \leq i_L} w_{i_1 \cdots i_L} x_{i_1}^{(j)} \cdots x_{i_L}^{(j)}$$

and

$$\bar{f} = \frac{1}{N} \sum_j f(u_j)$$

We have considered so far the two-class case. The extension to the multi-class case is as follows. We apply the previously described procedure (preferrably the curved boundary one), i.e. partition the patterns into two groups. One group of patterns S^+ corresponds to positive outputs and represents an estimate of a collection of classes. The other group S^- corresponds to negative outputs and represents an estimate of the remaining classes. Then we consider the group S^+ separately (i.e. we use only the patterns in S^+ for training), and partition it further into two groups S^{++} and S^{+-} corresponding to positive and negative outputs respectively. Similarly we partition S^- into S^{-+} and S^{--}. We continue in this hierarchical manner until we end up with the final classifier, see Fig. III.3 for an example. We stop the partitioning of a group whenever the criterion function J associated with the partition falls below a certain threshold. (One practically observes that J in general decreases slightly after each partition until one cluster remains whose partitioning results in a relatively abrupt decrease of J.) The final classifier consists now of several neurons, the sign pattern of whose outputs is an encoding of the class estimate of the presented pattern. Note that the output of the neuron responsible for breaking up some group plays a role in the encoding of only the classes contained in that group. Therefore, the encoding of a class could have a number of don't cares. Refer to the next section for a constructive example of the extension to the multi-class case.

III.4 IMPLEMENTATION EXAMPLES

The new method is implemented on two two-class problems and a five-class problem. In all examples the patterns of each class are generated from bivariate Gaussian distributions. The first example is a two-class problem with a large overlap between the two classes. Fig. III.4 shows the resulting decision boundary when applying the new method using a first-order neuron. One observes that the obtained partition agrees to a large extent to that a human would estimate when attempting to decompose the two clusters visually. Regarding the second example, we have two classes with equal diagonal covariance matrices whose diagonal entries differ much, resulting in two long clusters. Fig. III.5 shows the results of the application of the proposed method using a first-order neuron. In the last example we have a five-class problem with the means being on the corners and the center of a square. We used third-order neurons. Fig. III.6 shows the results. We ended up with four neurons partitioning the patterns. The first neuron is responsible for partitioning the whole collection of patterns into the two groups S^+ and S^-, separated by the boundary B_1; the second neuron partitions the group S^+ into the groups S^{++} and S^{+-}, separated by the boundary B_2; the third neuron divides S^{++} into S^{+++} and S^{++-} with boundary

B_3; finally the fourth neuron partitions S^- into S^{-+} and S^{--}, the boundary being B_4 (see also Fig. III.3). One observes that the method partitioned the patterns successfully. As a pattern classifier, the four neurons give the encoding of the class estimate of the presented pattern. The codes of the five classes are $+++X$, $++-X$, $+-XX$, $-XX+$, and $-XX-$, where the $+$'s and $-$'s represent the signs of the outputs and the X means don't care.

III.5 CONCLUSIONS

A new neural unsupervised learning technique has been proposed in this work. This technique is based on the hierarchical partition of the patterns. Each partition corresponds to one neuron, which is in general a higher-order neuron. The partition is performed by iterating the neuron weights in an attempt to maximize a defined criterion function. The method is implemented on several examples and is found to give good results. In the second implemented example (Fig. III.5) the method obtained a good solution, whereas the traditional adaptive resonance method [2] and self-organizing maps [3] produced unsatisfactory results. The method is fast, as it takes typically from about 2 to 5 iterations to converge. Although the proposed method is prone to get stuck in local minima, this did happen in our simulations in only very difficult problems and this problem could be solved by using gradient algorithms for searching for the global maximum, like the Tunneling Algorithm [8].

REFERENCES

[1] R. Duda and P. Hart, *Pattern Classification and Scene Analysis*, John Wiley and Sons, New York, 1973.

[2] S. Grossberg, "Adaptive pattern classification and universal recording: Part I. Parallel development and coding of neural feature detectors," *Biological Cybernetics, 23*, pp. 121-134, 1976.

[3] T. Kohonen, *Self-Organization and Associative Memory*, Springer-Verlag, Berlin, 1984.

[4] A. Atiya, "A neural unsupervised learning technique," *Neural Networks*, Vol. 1, Sup.1, p.69, Abstracts of the First Ann. INNS Meeting, Boston, MA, 1988.

[5] D. Psaltis and C. Park, "Nonlinear discriminant functions and associative memories," J. Denker, Ed., AIP Conf. Proc., Snowbird, Utah, 1986.

[6] D. Psaltis, C. Park and J. Hong, "Higher order associative memories and their optical implementations," *Neural Networks*, Vol. 1, No. 2, pp. 149-163, 1988.

[7] C. Giles and T. Maxwell, "Learning, invariance, and generalization in high-order neural networks," *Applied Optics*, Vol. 26, No. 23, pp. 4972-4978, 1987.

[8] A. Levy and A. Montalvo, "The Tunneling Algorithm for the global minimization of functions," *SIAM J. Sci. Stat. Comput.*, Vol. 6, pp. 15-29, January 1985.

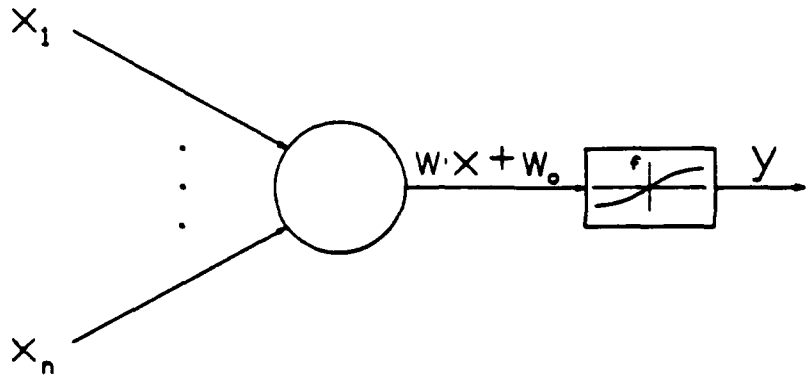

Fig. III.1: The model of a first order neuron with a sigmoid-shaped function.

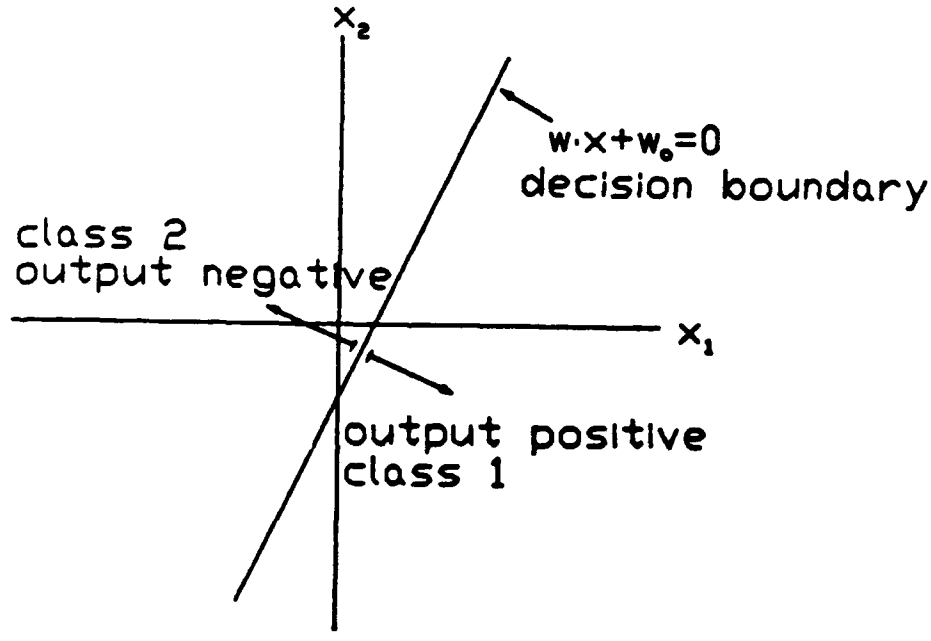

Fig. III.2: The decision regions and the decision boundary of a first order neuron example.

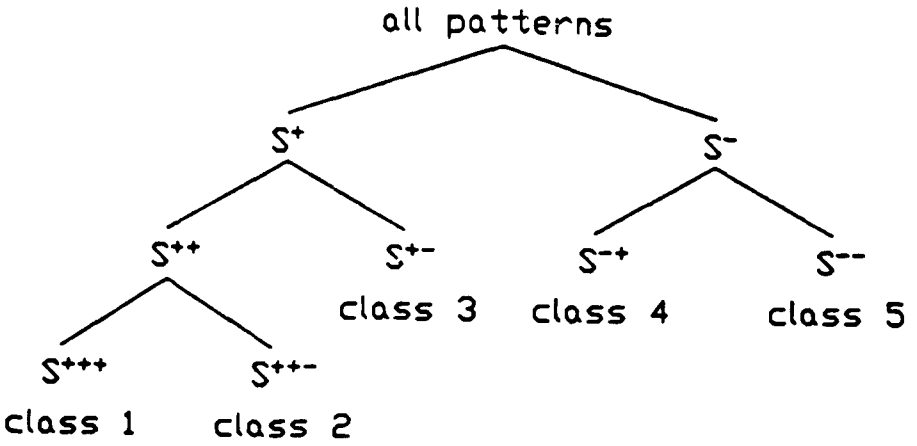

Fig. III.3: An example of the hierarchical design of the classifier.

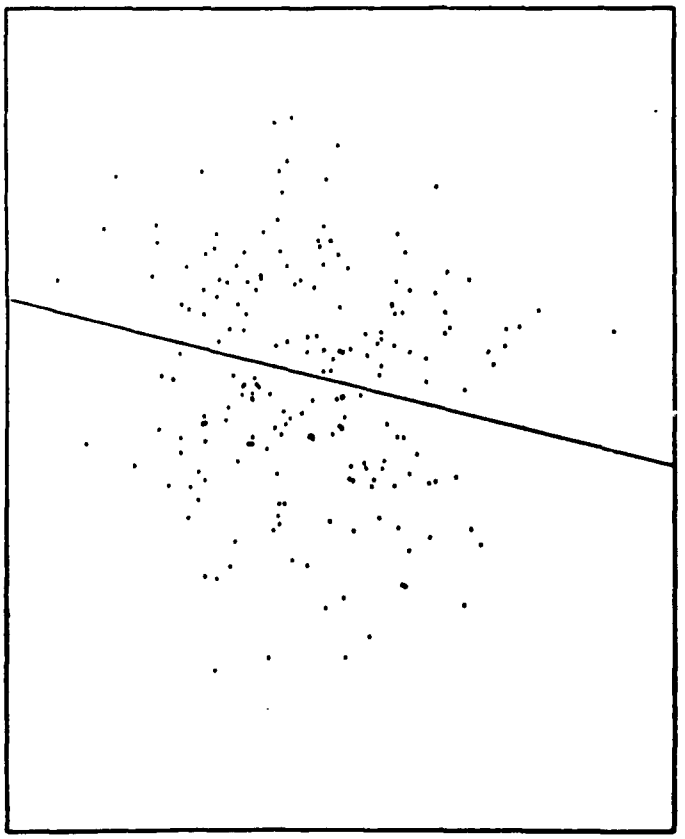

Fig. III.4: A two-cluster example with the decision boundary of the designed classifier.

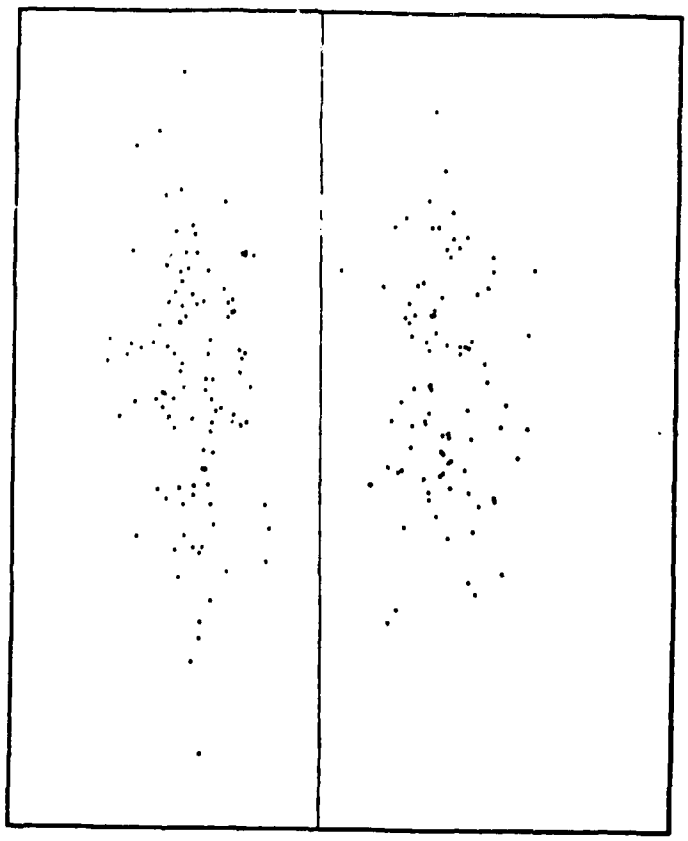

Fig. III.5: A two-cluster example with the decision boundary of the designed classifier.

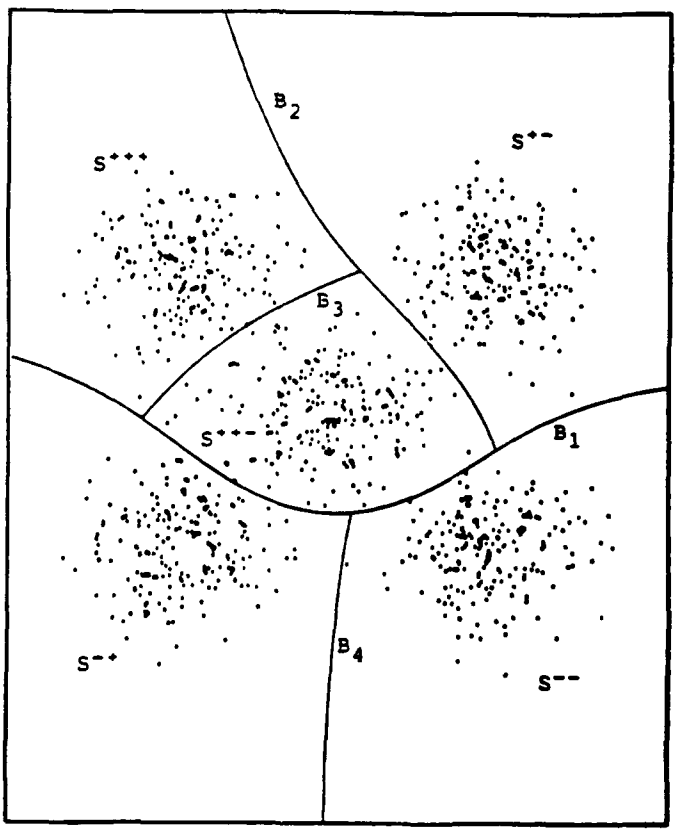

Fig. III.6: **A five-class example showing the decision boundaries of the hierarchically designed classifier. There are four neurons associated with the boundaries B_1, B_2, B_3 and B_4.**